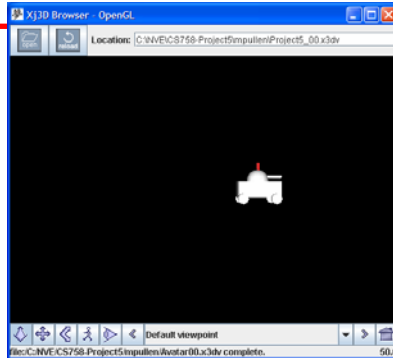


# Project 5 Introduction

---



The author of these slides is Dr. Mark Pullen. Students registered Dr. Pullen's course may make a single machine-readable copy and print a single copy of each slide for their own reference, so long as each slide contains the copyright statement. Permission for any other use, either in machine-readable or printed form, must be obtained from the author in writing.

# Project Purpose

---

- The purpose of this project is to learn about 3D support of collision detection by programming an avatar to change its appearance on receipt of a multicast DIS collision PDU.

## Ground Rules

---

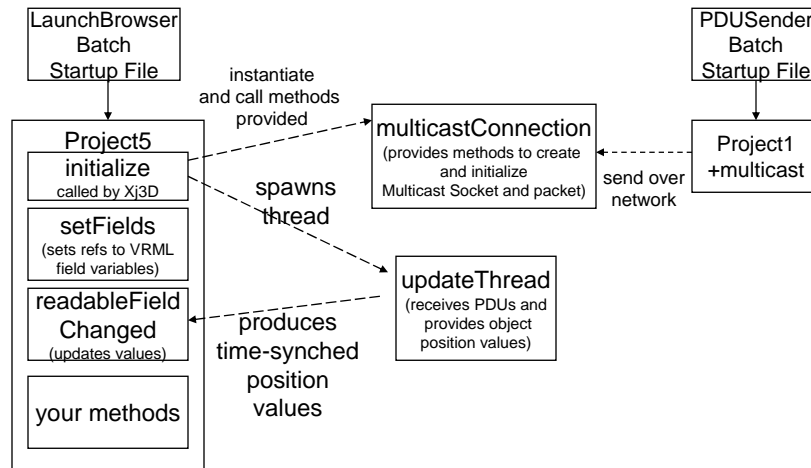
This project is to be done individually. Help as needed is OK, and should be acknowledged in your submission, but no team efforts in the first six projects please.

## Re-Use of Previous Projects

---

- Network code for this project is the same as Project 4, but modified for multicasting.
- VRML for this project is the same as Project 4, modified to cause a visible change in the avatar when a DIS Collision PDU is received.

## System Diagram



CS758-2-P5 SPRING 07

4/21/2007

© 2007 J. Mark Pullen

5

## Assignment

- Add to your VRML avatar some feature that changes in a very obvious way.
  - See VRML information which follows.
  - The “B” project will change *diffuseColor* in the material of some visible part
- Convert your Project1 and Project4 to multicast.
  - See Java information which follows.
- To demonstrate:
  - Run your avatar in Xj3D.
  - Run CollisionPDUGenerator, available from the course website.
- To earn an A, make the avatar change more appropriate to a collision.
  - \* your computer will need to have an active LAN connection

CS758-2-P5 SPRING 07

4/21/2007

© 2007 J. Mark Pullen

6

## VRML for Collision Indication

---

- In the final class we'll be assembling all of our avatars into one big .x3dv file
  - To avoid confusion, each student will receive a unique number to be used in externally visible names
  - *nn* in the slides that follow
- We are going to add some optional VRML to be switched on and off by an external input
- And modify our updateThread to listen for collision PDUs
  - In multicast mode
- Due to a bug in Xj3D, we will have to code the shape and material that will have its *diffuseColor* (or any other property) changed in the main file Project5\_*nn*.x3dv

## Recall: VRML File Project3.x3dv

---

```
#X3D V3.0 utf8
PROFILE Immersive
DEF MoveAvatar Transform {
    # we'll put an extra "children" here to hold the new
    # shape and material and use them in the avatar
    # you should put anything that changes appearance here
    children [
        Inline {
            url [ "Project3-avatar.x3dv" ]
        }
    ]
}
DEF Mover Script {
    outputOnly SFVec3f CurrentPosition
    url [ "Project3.class" ]
}
ROUTE Mover.CurrentPosition TO MoveAvatar.set_translation
```

## Project5\_nn.x3dv Structure (1)

the object (see Project3)

```
#X3D V3.0 utf8
PROFILE Immersive
DEF MoveAvatarnn Transform {
  children[
    Transform{
      children[
        Shape {
          appearance Appearance {
            material DEF Materialnn Material {
              you pick this – be sure to include diffuseColor
              geometry and this { and this }
            }
          ]
          and the transformation: rotation, scale, translation...
        }
      ]
    }
  ]
  children [
    DEF Avatarnn Inline { url "avatarnn.x3dv" } # avatar that uses Materialnn
  ]
}
```

CS758-2-P5 SPRING 07 4/21/2007 © 2007 J. Mark Pullen 9

## Project5\_nn.x3dv Structure (2)

control information (see Project3)

```
DEF Movernn Script {
  inputOnly SFTime TimeStep
  outputOnly SFColor CollisionColor
  outputOnly SFVec3f CurrentPosition
  url [ "Project5_nn.class" ]
}
DEF Steppernn TimeSensor {
  cycleInterval you provide this
  loop TRUE
}
ROUTE Movernn.CollisionColor TO Materialnn.set_diffuseColor
ROUTE Steppernn.cycleTime TO Movernn.TimeStep
ROUTE Movernn.CurrentPosition TO MoveAvatarnn.set_translation
```

## Recall:Project4.java (1)

### sending signals to VRML

---

```
// format of Java (code omitted at ...)
import java.util.Map;
import org.web3d.x3d.sai.*;
import java.io.*;
import java.net.*;
public class Project4 implements X3DScriptImplementation,
    X3DFieldEventListener
{
    private startTime = System.currentTimeMillis();
    private SFVec3f currentPosition;
    private SFTime timestep;
    private Object syncObject = new Object();
    private DatagramSocket socket;
    private DatagramPacket packet;
    private byte[] bytePDU = new byte[144];
    private int xPosition; // shared variable
    ...

```

CS758-2-P5 SPRING 07

4/21/2007

© 2007 J. Mark Pullen

11

## Project4.java (2)

---

```
public void setFields(X3DScriptNode externalView, Map fields)
{
    // Links to our object and timer in VRML scene
    currentPosition = (SFVec3f)fields.get("CurrentPosition");
    timestep = (SFTime)fields.get("TimeStep");
    timestep.addX3DEventListener(this);
}
public void initialize()
{
    // initialize shared variables
    ...
    // initialize socket and packet using methods in
    // loopbackConnection internal class
    socket = loopback.createSocket();
    packet = loopback.createPacket();
    // spawn updateThread to receive PDUs
    ...
}

```

CS758-2-P5 SPRING 07

4/21/2007

© 2007 J. Mark Pullen

12

## Project4.java (3)

---

```
public void readableFieldChanged(X3DFieldEvent evt)
{
    // called by VRML framework when TimeStep changes
    long currentTime = System.currentTimeMillis();
    float deltaT = (float)(currentTime - receivedPDUTime)*.001f;

    // compute dead reckoning
    float xUpdate, yUpdate, zUpdate;
    xUpdate = ...

    // apply dead reckoning
    synchronized(syncObject)
    {
        updateObject(xUpdate, ...); // you provide this method
    }
} // end readableFieldChanged
```

## Project4.java (4)

---

```
// internal class to implement thread that provides the position values
public class updateThread implements Runnable
{
    public void run()
    {
        while(true)
        {
            // read the PDU and extract object parameters
            loopback.readPacket(packet);
            synchronized(syncObject)
            {
                extractDataFromPDU();
            }
        } // end while(true)
    } // end run() method
}
```

## Project4.java (5)

---

```
public void extractDataFromPDU()
{
    byte[ ] buffer = new byte[8];
    // extract x,y,z locations and velocities
    // x Location
    System.arraycopy(packet.getData(),48,buffer,0,8);
    xLocation = decodeDouble(buffer);
    ...
} // end of extractDataFromPDU()
// helper classes
} // end class updateThread
```

## Controlling the Collision Color (1)

code omitted at ...

---

```
public class Project5_00 implements X3DScriptImplementation,
    X3DFieldEventListener
{ ...
    private float[] changeableColor = {0,0,1}, red = {1,0,0};
    boolean collisionOccurred = false; // set this on sensing collision
    ...
    // getting reference to the color variable from VRML
    public void setFields(X3DScriptNode externalView, Map fields) {
        ...
        changeableColor = (SFColor)fields.get("CollisionColor");
        chanegableColor.setValue(blue);
    }
    ...
    // setting the color
    public void readableFieldChanged(X3DFieldEvent evt) {
        ...
        if(collisionOccurred) changeableColor.setValue(red);
    }
}
```

## Controlling the Collision Color (2)

code omitted at ...

---

```
// for now, simple test for collision PDU
public class updateThread implements Runnable {
    ...
    public void extractDataFromPDU() {
        ...
        if((packet.getData)[2] == 4)
            collisionOccurred = true; // collision PDU code    ...
        }
    }
}
```

## Java for Multicast (1)

---

```
// unicast sender (assuming byte[ ] buffer exists)
InetAddress address;
int portNumber;
try
{
    address = InetAddress.getByName("131.120.7.12");
}
catch(UnknownHostException uhe)
{
}
DatagramSocket socket;
DatagramPacket message = new DatagramPacket(buffer,
    buffer.length, address, portNumber);
```

## Java for Multicast (2)

---

```
// more unicast sender
try
{
    socket = new DatagramSocket( );
}
catch(SocketException se){}
try
{
    socket.send(message);
}
catch(IOException ioe) { }
```

## Java for Multicast (3)

---

```
// multicast sender (assuming byte[ ] buffer exists):
InetAddress address=null;
int portNumber;
try
{
    address = InetAddress.getByName("225.0.0.1");
}
catch(UnknownHostException uhe) { }
MulticastSocket socket;
DatagramPacket message = new DatagramPacket(buffer,
    buffer.length, address, portNumber);
```

## Java for Multicast (4)

---

```
// more multicast sender
try
{
    socket = new MulticastSocket( );
}
catch(IOException ioe) { }
try
{
    socket.send(message);
}
catch(IOException ioe) { }
```

## Java for Multicast (5)

---

```
// unicast receiver (assuming byte[ ] buffer exists):
DatagramSocket socket;
int portNumber = 7777;
DatagramPacket message =
    new DatagramPacket(buffer, buffer.length);
try
{
    socket = new DatagramSocket(portNumber);
}
catch(SocketException se){ }
try
{
    socket.receive(message);
}
catch(IOException ioe) { }
```

## Java for Multicast (6)

---

```
// multicast receiver (assuming byte[ ] buffer exists):
// unlike unicast case, must join the multicast group
InetAddress address=null;
int portNumber = 7777;
try
{
    address = InetAddress.getByName("225.0.0.1");
}
catch(UnknownHostException uhe) { }
MulticastSocket socket;
DatagramPacket message =
    new DatagramPacket(buffer, buffer.length);
```

## Java for Multicast (7)

---

```
// more multicast receiver
try
{
    socket = new MulticastSocket(portNumber);
    socket.joinGroup(address);
    socket.send(message);
}
catch(IOException ioe) { }
```

## Converting to Multicast

---

- Modify Project1 to send multicast
- Modify Project4 to receive multicast
- Rename “loopbackConnection” internal class to “multicastConnection”
  - Add a new method “joinGroup” (invoke it in initialize() )

```
public void joinGroup(String groupAddress)
{
    // logic to join group
}
```
  - Modify createSocket to create a multicast socket.
  - Use this internal class to instantiate network objects

## Testing

---

- Windows multicast networking requires active connection to LAN
  - Wireless LAN works
  - So does a hub with nothing else connected(!)
- Run your code to confirm the avatar moves as before
- To test the collision indicator, download CollisionPDUGenerator.java from the course website
  - Run it and enter 0
  - Color of Material should change
  - We’ll use the Entity ID in Project 6

## How to Submit

---

- Submit your project in an email to the instructor with subject line containing the course number and "Project 5".
- Send an attachment zipfile containing all .x3dv, .java, and .class files, README file as before.
- In Project 6 you will detect collision using this avatar and expanded network reader
- For final session, instructor will assemble VRML files from all students
  - Be sure to use your two-digit number in all names!